NASA Technical Memorandum 102721

(NASA-TM-102721) GCS PROGRAMMER'S MANUAL N91-17612
(NASA) 28 p CSCL 09B

Unclas
G3/61 0330163

GCS PROGRAMMER'S MANUAL

Douglas S. Lowman, B. Edward Withers,
Anita M. Shagnea, Leslie A. Dent, and
Kelly J. Hayhurst

December 1990

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Preface

The *GCS Programmer's Manual* is Document No. 4 in a series of 15 documents which fulfill the Radio Technical Commission for Aeronautics RTCA/DO-178A guidelines, "Software Considerations in Airborne Systems and Equipment Certification."[1] The documents are numbered as specified in the DO-178A guidelines. The documents in the series are used to demonstrate compliance with the DO-178A guidelines by describing the application of the procedures and techniques used during the development of flight software. These documents were prepared under contract with NASA Langley Research Center as a part of their long-term research program addressing the fundamentals of the software failure process.

This project consists of two complementary goals: first, to develop software for use by the Research Triangle Institute (RTI) in the software error studies research program sponsored by NASA Langley Research Center [2]; second, to use and assess the RTCA/DO-178A guidelines for the Federal Aviation Administration (FAA). The two goals are complementary in that the use of the structured DO-178A guidelines in the development of the software will ensure that the test specimens of software have been developed according to the industry standards for flight-critical software. The error studies research analyses will then be conducted using high-quality software specimens.

The implementations will be subjected to two different software testing environments: verification of each implementation according to the RTCA/DO-178A guidelines and replicated random testing in a configuration which runs more than one test specimen at a time. The term *implementations* refers to bodies of code written by different programmers, while a *version* is a piece of code at a particular state (i.e., Version 2.0 is the result of code review). This research effort involves the gathering of product and process data from every phase of software development for later analysis. More information on the goals of the Guidance and Control Software (GCS) project are available in the *GCS Plan for Software Aspects of Certification.*

The series consists of the following documents:

- *GCS Configuration Index* Document No. 1
- *GCS Development Specification* Document No. 2
- *GCS Design Descriptions* One for each software implementation. Document No. 3
- *GCS Programmer's Manual* Document No. 4, includes Software Design Standards, Document No. 12.
- *GCS Configuration Management Plan* Document No. 5A
- *Software Quality Assurance Plan for GCS* Document No. 5B
- *GCS Source Listing* One for each software implementation. Document No. 6
- *GCS Source Code* One for each software implementation. Document No. 7

- *GCS Executable Object Code* One for each software implementation. Not available on hardcopy. Document No. 8
- *GCS Support/Development System Configuration Description* Document No. 9
- *GCS Accomplishment Summary* Document No. 10
- *Software Verification Plan for GCS* Document No. 11
- *GCS Development Specification Review Description* Document No. 11A
- *GCS Simulator (GCS_SIM) Verification Plan and System Description* Document No. 13
- *GCS Plan for Software Aspects of Certification* Document No. 14

Contents

Foreword

This document is the result of the compilation of a variety of programmer instructions which were developed to help further describe various phases of the Guidance and Control Software (GCS) project as suggested by the RTCA/DO-178A guidelines. As required by the DO-178A, this document describes additional requirements or instructions in the form of a programmer's instruction manual. Each of the sections in this document describes a separate programmer instruction. The author(s) of each section is listed in Table 0.1.

Table 0.1. Author(s) of each Programmer Instruction

Number	Title	Author
1	Activity Recording	Douglas S. Lowman
2	Communication Protocol	Douglas S. Lowman
3	Coding Standards for GCS Applications	B. Edward Withers
4	Change Management	B. Edward Withers
5	Error Handlers	Douglas S. Lowman
6	Design Standards	B. Edward Withers
7	Design Documentation Outline	B. Edward Withers
8	Completing the Problem Report Form	Kelly J. Hayhurst & Leslie A. Dent
9	Using the GCS Module Test Log	Leslie A. Dent
10	New Formats for Documentation	Anita M. Shagnea & Douglas S. Lowman
11	Accuracy Requirements	Anita M. Shagnea
12	Programmer Responsibilities	Anita M. Shagnea

Document No. 12, *Software Design Standards*, required by the RTCA/DO-178A guidelines, has been included in this document as one of the programmer instructions (See Chapter 7).

This document will be redistributed and the release number incremented any time there is a change to the document — this includes the creation of additional programmer instructions or requirements.

Note: Instructions in this document **supercede** all programmer instructions distributed prior to the adoption of the DO-178A guidelines. Due to the fact the GCS project is targeted for VAX/VMS systems, all instructions in this document assume a prior knowledge of the VAX/VMS operating system. Descriptions of software tools referenced in this document can be found in the *GCS Support/Development System Configuration Description* document.

1. Activity Recording

This chapter describes the various GCS programmer activity recording requirements. Activity recording will help in the collection of programmer effort data required by the project.

Activity recording is broken down into three separate areas. These areas include:

- software problem reports,
- weekly status reports, and
- engineering activity log.

Error and change management information will be recorded in the form of Software Problem Reports (PRs) which are explained in detail in Section 8 of this document. (Note: PRs will not be used until the programmer's first design review.)

Weekly status reports will give the project management team an idea of the progress of a particular programmer and perhaps alert them to unresolved problems that may impact scheduled milestones. The weekly status reports will be contained in a VAX Notes conference, "WEEKLY", which has been created (by the project management team) and exists in the project GCS_NOTES subdirectory. Each weekly status report should be in the form of a note and will be in the following format:

Introduction — An introduction covering the objectives and scope of the work effort for the previous week.

Technical Progress Summary — A description of the overall progress since the last status report. Descriptions shall include pertinent data to detail and explain any significant results achieved. A summary of progress, data and conclusions for significant completed phases of the work shall be included, when needed, for comprehension of the total progress to date.

Current Problems — A description of current problems that may impede performance, along with proposed corrective action. Such problems include technical and schedule status.

The "Engineering Notebook" is also a VAX Notes conference in the project GCS_NOTES subdirectory. This notebook is an on-line diary of thoughts, ideas, and work completed. Information in this notebook may be included in the weekly status reports. This notebook will not have a strict format; it can be organized and structured or completely unstructured. From time to time someone from the project management team may review a programmer's "Engineering Notebook" to see how things are going.

2. Communication Protocol

This chapter describes how general information will be distributed to each GCS programmer and how he may ask questions about the GCS specification, the simulator test-harness, or RTI programming environment.

VAX Notes will be used to help manage programmer questions and post general information. It also offers the advantage of keeping a detailed record of all questions and responses which can be searched by a variety of criteria, such as programmer name, subject, or keyword.

In addition to the VAX Notes conferences mentioned in Section 1, each programmer will have a "PROBLEM" conference where he may submit any questions, comments, and problem resolution recommendations that relate to the GCS specification.

The programmers involved in this experiment will *not* be able to read information in personal conferences other than their own. The GCS project management team will be reviewing questions "posted" to each personal conference and "replying" to each question by posting a response to the same personal conference.

In order to maintain a standard reporting method, the following format for posting questions to the programmer "PROBLEM" conferences is to be used:

```
PROGRAMMER ID :
SUBJECT   :
AUDIENCE : User Specific or General

QUESTION :
   .
   .
   .

RECOMMENDATION(S):
   .
   .
   .
```

If a question is deemed general by the project team, a response will be posted to the GCS GENERAL INFORMATION "electronic bulletin board". Only the RTI project management team will be able to post new items to this bulletin board. Programmers should read this bulletin board, but unlike their personal conferences, they will not be able to post or reply to items on this general bulletin board. This restriction helps assure that programmers do not accidently post a question to a general bulletin board where it would be seen by other programmers.

Following is a copy of the welcome message that has been posted to the GCS General Information Bulletin Board, describing that bulletin board in a little more detail.

```
                -< GCS General Information Bulletin Board >-
================================================================================
Note 1.0                General Information              No replies
DSL                           37 lines  12-APR-1988 18:32
--------------------------------------------------------------------------------

Welcome to the Guidance and Control Software (GCS) general information
bulletin board.  The purpose of this bulletin board is to provide
a facility for distributing general information to all GCS programmers.

Information found in this bulletin board may include the following:

* Additional program instructions

* Questions asked by other programmers that require general answers

* General answers to questions (when appropriate)

* Other project development information
   (eg. program implementation procedures, project schedule changes, etc.)

This bulletin board is divided into four topic areas.  Each of the
topic areas will contain information relating only to that topic.

The following topics should be read by all GCS programmers:

* General Information
* GCS Specification Questions and Answers
* GCS_SIMulator Questions and Answers
* RTI Programming Environment  (VMS and Tool Question and Answers)

If you would like to respond to something you have read on this
bulletin board, post a question to your "personal" conference
and the Project Team will respond to your comment either to you
individually or will post your comments and our responses to this
board.

================================================================================
[End of Note]
```

Since we are using VAX Notes as a means for keeping track of questions and comments relating to the development of GCS program versions, we would like to restrict the use of the VAX/VMS Personal Mail Utility (MAIL).

In compliance with the project guidelines, GCS programmers should not refer to the GCS experiment in any way when communicating with other GCS programmers. Mail messages to the GCS project management team relating to project status information, etc., are permitted, but should not include questions relating to the GCS specification.

3. Coding Standards for GCS Applications

This chapter contains modifications to the VMS FORTRAN Code Generation Guidelines[3] that the programmers are required to use. Information contained in this section supercedes specific sections of the code generation guidelines and applies only for code used in GCS applications.

Module Header Block — The "ERRORS" field in the module header should be changed to "ERROR HANDLING", and should contain information about any embedded error handling that the programmer has included that is not previously specified.

Revision History — The revision history should contain the Problem Report (PR) number associated with the change made during each version. Notation of changes by version number should begin at the time of the design review of the application; therefore, there should always be an associated PR, but only one PR for each problem. Note that there are Revision Histories in both the Module Header Block and the Include File Header Block.

Notation of Modifications — Modifications to code should be noted with a version number as specified in the Guidelines. Also, the beginning of all areas of changes should be noted with a comment line containing the following:

```
!+
!   Begin changes PR#<prnumber> Vn
!-
```

The end of change areas should be similarly marked by an "End Change" comment line.

4. Change Management

Change management for each implementation begins at the time of the programmer's design review.[1] The team*work*[2] design diagrams created by each programmer will be placed under configuration management as will the code generated by each programmer. The team*work* diagrams will be printed for the design review, and these hardcopies will be kept by the configuration manager (B. Edward Withers) or the SQA representative (Stephen E. Duncan) for later use in tracing changes in design and tracing requirements from design to code. Beginning at the time of the design review, a problem report must be generated for each change that is required. When the appropriate fix has been put into place, the SQA representative must approve the change and certify that the change specified in the problem report (and only that change) has been made to the design. To provide configuration management, each problem report that requires a change to the design should be accompanied by a printout of all pages of the design that have been modified.

The programmer's code will be placed under configuration management at the time of the code review. Again, problem reports will be generated at the code review and throughout the testing of the code. Some of these problem reports will only cause changes in the code, while others will cause changes in the design as well.

Changes to code will require the programmer to request that the SQA representative or the configuration manager reserve an official copy of his code for him to fix. When the code has been fixed, the SQA representative must certify that the requested changes have been made, then he may replace the corrected version of code in its library. (Note: the SQA representative has full power to refuse to replace code if he feels that the problem has not been correctly fixed, or if he feels that undocumented changes have been made.)

Each programmer will test his own version during the unit testing phase; thus, some of the above rules will be relaxed somewhat.[3] At the beginning of unit testing, a complete copy of the programmer's code will be reserved and given to the programmer. During unit testing, when a problem is encountered, the programmer should generate a problem report and fix the problem in the code that he is using. When unit testing is finished, the SQA representative will review the problem reports generated and the code that the programmer submits to be replaced into configuration control. This should speed up the turnaround time on changes made at this level of testing and reduce the overhead required.

---

[1] See the *GCS Configuration Management Plan* for details.

[2] Team*work* is a registered trademark of Cadre Technologies Inc.

[3] See the *Software Verification Plan for GCS* for details.

5. Error Handlers

GCS programmers may write very limited error handlers if they choose. The writing of error handlers is **not** required by the GCS specification; however, since the goal is to write reliable software, the use of error handlers may be desirable.

Since this experiment is tasked with studying errors discovered in independently developed Guidance and Control Software implementations, the experiment guidelines listed below are more restrictive and, in some cases, are in direct conflict with critical-system error recovery techniques. (For example, by preventing the use of VMS system services and/or user-written assembler routines, the GCS implementations cannot possibly recover from an arithmetic operation triggered overflow or underflow exception.)

If it is decided by a GCS programmer that an error recovery routine should be written, the following guidelines **must** be observed:

Guidelines

- GCS Implementations must have the ability to execute with **and** without the error handler(s) enabled. GCS implementations using error handlers should read the error handler on/off setting information out of a data file. (The filename should include the programmer's PLANET name as part of the file prefix.)
- Do not use error handlers in lieu of rethinking algorithms.
- Use error handlers only if they are natural to your programming style.
- Remember, error handling routines contribute to overall program complexity, so use error handlers wisely.
- Follow system routine restrictions defined in the Implementation Notes Section of the *GCS Development Specification.*
- Error handlers may be developed for single version testing; multiple paths and parallel execution streams are not allowed.

Mechanics

- An informational message about anomalous behavior of code should be written out to FORTRAN logical unit 6 (FOR006). (You do not need to define a data file to catch the output; it is done automatically for you by GCS_SIM.)
- Error handlers may **only** be written using VAX/VMS standard FORTRAN.
- No ASSEMBLER routines or DEC supplied library routines may be used.
- Errors can only be handled in a preventive manner. Error handlers may not intercept system fatal errors; once the program fails at the system level, it fails.

It is important to note that this instruction *overrides* the section in the VMS FORTRAN Code Generation Guidelines[3, Pages 11 and 12].

6. Design Standards

Following are the design requirements for the GCS experiment.

The first requirement deals with the method of design to be used. Structured analysis or structured design as described by Derek Hatley[4], Tom DeMarco[5] and Paul Ward and Steven Mellor[6] should be used for the design of the applications.

The second requirement is the use of a Computer Aided Software Engineering (CASE) tool, team*work*. This tool is to be used to aid in the structured design of the applications, and certain parts of the output from team*work* will be required for design and code reviews. Team*work* is composed of several tools that are available to the designer. The components of team*work* include, but are not limited to, the following components:

SA — The base-line structured analysis tool,

RT — An extension of SA that allows description of real-time systems, and

SD — A parallel tool that follows the Ward and Mellor approach.

It should be understood that the designer is free to use any of these tools as he sees fit, provided at least one of them is used.

Use of team*work* is further restricted in that:

- Process Specifications (P-Specs) and Module Specifications (M-Specs) should not be greater than two pages in length when printed.
- P-Specs and M-Specs should not contain comments to describe algorithms. (Pseudo-code should describe algorithms, but comments may be used to describe notation. Alternately, English may be used to describe the algorithms without using pseudo-code.) Note that pseudo-code is allowed in P-Specs for this experiment despite Hatley's recommendation that pseudo-code not be located in P-Specs.
- The lists of input and output variables should be directly traceable to the specification. (Any flows should be broken down to the elements as shown in the *GCS Development Specification* before entering the P-Spec or M-Spec.)
- Before printing the hardcopy that will be delivered at the design review, a complete "balance" check should be conducted on the model, and no changes should be made to the model between the *last* "balance" and the print.

7. Design Documentation Outline

This chapter outlines the contents of the programmer design documentation. The document produced from this outline will be used for the design review and may also be used later to trace changes to design. This document may also prove to be useful to the programmer as well as the testing team while debugging the code.

Below is a copy of a design document outline. Note that this is a SUGGESTED outline and may be rearranged or modified by the programmer as desired. However, all points in the outline listed below MUST be addressed in the final design document.

I Introduction to *Name of your planet*

a) **Top Level Description**
Should give brief overview of the context of the code (e.g., simulates the on-board navigational code for a planetary lander, etc.). This should also give a brief view of the organization of your code.

b) **Comments on Method**
This subsection should contain any comments that the programmer feels are needed in the context of the philosophy or methods used during the design of the software. Mention should be made of the tools used during design such as team*work*/SA, [1] team*work*/SD, etc.

II Program Structure

a) **Description of Program Structure**
This subsection should contain a design structure description of the software and should be organized with respect to the planned modules to be written, not necessarily the organization of the *GCS Development Specification.* This description may be contained within the data/control flow diagrams, or may be some type of list that shows the call structure of the code.

b) **Module Description**
This subsection should contain a **BRIEF** description of each code module. (Again this means the modules envisioned for the code and not the modules specified in the *GCS Development Specification.*) These descriptions should map on a one-to-one basis with the modules in the team*work* diagrams and should be limited to one or two line descriptions. This description may include information about design modules that may be combined into larger code modules.

III Data and Control Flow

This section should include the structured analysis/structured design diagrams, and it may be that the design and control diagrams are actually combined into single diagrams for each level.

[1] Team*work* is a registered trademark of Cadre Technologies Inc.

a) **Data Flow Description**
This subsection should contain data flow diagrams as well as any explanation that the programmer feels is needed.

b) **Control Flow Description**
This subsection should contain control flow diagrams as well as any explanation that the programmer feels is needed.

IV **Coding Construction Notes**
Note that some of the subsections listed may not be needed and thus are optional. Mention should be made of things such as exception handling, use of CMS, etc.

a) **Algorithms Not Specified**
This subsection should contain descriptions of any algorithms used that were not supplied in the *GCS Development Specification*. This does not imply that the algorithms in the specification are to be ignored, only that there are some algorithms that were left to the programmer's discretion.

b) **Program Interrupts**
This subsection should detail any program interrupts that were not required by the *GCS Development Specification* but were placed into the code.

c) **Tools Used to Construct/Manage Code**
This subsection should address any tools to be used to construct or manage the code.

d) **Memory Size and Organization**
This subsection should describe the size and organization of any memory used by the application that was not required by the GCS application.

e) **Data Dictionary**
This subsection should contain a complete data dictionary as used by the programmer, including both specified and non-specified variables. Note this subsection may contain all the information required for memory size and organization, thus negating the need for that subsection. This subsection is **NOT** optional.

f) **References**
References used for the design/construction of the code should be listed here. This may take the form of a bibliography.

The design document should follow a format loosely similar to that of the *GCS Development Specification* and/or the Hatley book on real-time system specification[4].

The programmer should supply a snapshot of team*work* files at the time of the design review. The design document may reference parts of this snapshot rather than actually including them; however, all such references should be clearly defined and easy to locate. A hardcopy placed into a binder with sections clearly marked would be helpful.

The reference section of the design document should include a reference to the *GCS Development Specification*, and if there is any information or lack of information from the specification in the design document, this should be clearly marked.

It is important to note that all documentation should reference the *planetary name* of the version, but not directly reference the name of the programmer.

8. Completing the Problem Report Form

The GCS Problem Report (PR) Form (See the *Software Verification Plan for GCS*) is to be used to document any changes made during the GCS project. The primary objective of the PR form is to capture as much data as possible concerning all changes made after formal verification has started in the project. The first PR forms for an implementation will be filled out at the design review. Changes to the GCS Development Specification, the GCS Design, the GCS code, and GCS test cases must be documented. It is important to have a detailed description of each change and its corresponding fix so that any given stage of the experiment can be recreated.

When a participant in the experiment (programmer, tester, SQA representative, or User/Analyst) discovers a change in some part of the experiment that he feels should be made, it is that person's responsibility to initiate a PR form. The following procedure should be followed.

1. The initiator of the PR form fills out the form through the section entitled "Explanation of Fault/Error Detection."

2. The initiator gives the PR form to the person who will be making the change.[1] All GCS Specification changes go to the User/Analyst. All implementation specific changes go to the implementation programmer. All test case changes go to the implementation tester.

3. The person who is going to make the change obtains those items that need to be changed from the configuration management team for those items that need to be changed. He makes the changes[2] and fills out the PR form[3] through the section: "When did the error enter the system?"

4. The person who made the change discusses the change with the SQA representative.[4]

5. If the SQA representative approves the change, the changed item is resubmitted to configuration control.

6. The PR form is returned to the initiator.

7. The initiator checks to see that the proper change has been made and signs his approval or returns the form to the person who made the change to be changed again.

---

[1]Sometimes the initiator and the person who makes the change will be the same person. In those cases, it is still important that all of the information is captured on the PR form.

[2]If the change is in the code, it is documented as described in Section 3 of this document - *Coding Standards for GCS Applications.*

[3]If the change is in the design, a copy of the specific changed design diagram should be attached to the PR form.

[4]While the programmer is performing unit testing, the SQA representative does not need to approve changes, since the programmer handles his own configuration control during that stage. The SQA representative still has the final signature approval at the end of the cycle.

8. The PR form goes to the SQA representative. His signature under SQA approval completes the form and he keeps it.

The PR form itself is largely self-explanatory. Specific instructions or further explanation for some of the elements of the form are given below.

Page_of_ Be certain to fill out the relative page number on each form to help avoid the loss of any pages of the PR form.

PR No. The PR No. is sequential starting at 1 for each implementation. The SQA representative will keep track of the next available PR number.

Activity at Fault/Error Detection Time Check the appropriate box to describe the activity which was being conducted when the fault was detected. The *Reading Code*[5] box should be used by the programmer if a fault is noticed while another one is being fixed.

Tester Approval Signature given by the initiator of the PR form.

General Version Number The version number which is used for configuration control.

Effort Hours for Fix Documentation of the amount of time involved in debugging and correcting an error is important data, necessary for the analysis of the cost effectiveness of the formal testing process.

Description of Problem and Fix A general description is appropriate; there is no need to detail specific lines of code; citing them is sufficient.

Error Type Check the appropriate box using the descriptions[6] below as a guideline.

Computational Error - Any error in an equation, including:

- incorrect operand
- incorrect use of parenthesis
- unit or data conversion error
- missing computation

Control Flow Error - Any error which causes control to be passed to the wrong set of pseudo-code or statements; or causes processes, pseudo-code, or statements to be performed in the wrong order, including:

- incorrect operand in logical expression
- incorrect looping conditions

[5]During subframe testing, the programmer is prohibited from making any changes which are not directly related to an observed failure.

[6]These categories were adapted from the *RADC Software Test Handbook[7].*

- logic activities out of sequence

Data Handling Error - Any error which relates to manipulating files or variables, including:

- data not properly defined/dimensioned
- data initialization not done
- data written or read in wrong format
- incomplete or missing output
- data referenced out of bounds
- storage space exceeded

Interface Error - Any error which involves calling other processes, including:

- subroutine arguments not consistent in type, units, etc.
- nonexistent subroutine called
- synchronization error
- timing violation
- deadlock

Operation Error - Any error which is caused by something outside what is being tested, including:

- operating system error
- test case error
- simulator error

Inconsistency - Any error which is caused by contradictory parts in or between any GCS documents, including:

- inconsistent or ambiguous requirements
- code does not match design

Other - Any error which cannot be characterized by one of the above, including:

- incompatibility with project standards
- code or design inefficient
- extra functionality present

Briefly describe the error on the lines provided.

Error Severity The following definitions are given in DO-178A as guidelines for determining criticality categories:

Critical - A failure which would prevent the continued safe flight and landing of the spacecraft.

Serious - A failure which would cause incorrect results but allow the spacecraft to continue its flight.

Nonessential - A failure which would not significantly degrade spacecraft capability.

SQA Approval Signature of the SQA representative to be given after the change has been verified to be correctly made and documented.

Continuation of Section The final page of the problem report form is to be used if additional space is needed to record information. Identify the section which is being continued on the line provided. In the section itself, note that it is continued on another page.

9. Using the GCS Unit Test Log

The purpose of the GCS Unit Test Log[1] is to enable the programmer to keep track of the testing performed during unit testing. This instruction only describes how to use the test log. It is *not* intended as a guide to all of the unit testing activities. For a description of unit testing and a copy of the GCS Unit Test Log, see the *Software Verification Plan for GCS.*

The log consists of two different pages. Extra copies of either page should be added by the programmer as necessary. A new test log will be created for each unit. On each page of the test log the planet name, subframe, and unit name should be recorded in the appropriate place. On the first page of the log, the input to and output from the unit should be recorded. The input and output should consist of variables, with a brief definition if the name of the variable does not clearly indicate its purpose. Obviously since most GCS variables reside in the common regions of memory, the input and output for the unit will not be directly referenced in a unit call. For the sake of testing, the input and the output should be anything from the common regions of memory which the unit uses and/or modifies as well as variables which are passed in.

The second page of the GCS Unit Test Log is used to record test case information. The programmer should follow the procedure outlined below.

1. Give the test case a number or other form of identification.

2. While designing the test case, record the input variables and their values.

3. If there is an excessive number of inputs for a test case, it is permissible to indicate a file name in the input column instead of listing all the variables and their values. If the file method is chosen, a printout of the file should be attached to the log with the test case number clearly marked on it.[2]

4. Determine the expected results by recording the output variables and their expected values and expected accuracy where appropriate. The calculations required to determine the expected results should be included. Again, the file method may be chosen, with a printout of the file attached.

5. After all test cases are recorded and any test drivers are written, the tests are run and the date is recorded in the last column of the log.

6. Record the actual results of the test on the log, including the observed accuracy if appropriate. Again, the file method may be chosen, with a printout of the file attached.

7. If the actual results do not agree with the expected results (within the expected accuracy where appropriate), a GCS problem report form must be filled out and its number recorded in the first column of the log.

[1] After testing of the implementations began, the name of the Module Test phase was changed to Unit Test. Early GCS documents refer to a *Module* Test Log.

[2] Any files created during unit testing should be saved so that they may be placed under configuration control at the conclusion of unit testing.

8. If the problem was actually with the test case, one line should be drawn through that part of the test case and an arrow should point to the corrected part of the test case. A GCS problem report form does not need to be filled out.

9. After a problem is fixed and the test case is run again, the new actual results are recorded. If there was not space on the form, after the first execution of the test case, the results should be written at the end of the test form with the test case number serving to identify it.

10. When a correctly executed test case has to be run again, because of a change in the unit caused by another test case, the new run date should be added to the "date test executed" column. If the actual results of the rerun did not agree with the expected results, that information must be recorded on a separate line.

11. When all tests have been satisfactorily executed and the stopping rules met, the total number of test cases should be recorded on the first page of the log.

10. New Formats for Documentation

Each programmer is required to create a *GCS Design Description* and *GCS Source Listing.* In order to produce DO-178A documentation in a uniform format, a document driver template has been created and exists in a document template directory.

All documents are to be formatted using the document preparation system, LaTeX[8]. (Note:LaTeX user guides are available for reference.)

The format of the text of the Design Description should follow the format given in Section 6 – *Design Description Outline* of this document.

The document driver template for all GCS documents is in a directory referred to by the TEX_LOC: logical. This logical is automatically defined when a programmer executes the SETUP.COM file found in the project:[viking.gcs_tools] directory.

To obtain a copy of the document driver template, type the following command (where **planet** is the programmer's assigned planet name and **n** is the document number) at the VMS system prompt:

$ copy TEX_LOC:doc_template.tex [*directory*] *doc*_**planet**_**n**_*driver.tex*

After making a copy of the document driver template, edit the copy of the file and follow the instructions contained within the document driver template.

The cover sheet (report_cover.tex), the signature page (sigpage.tex), and the preface (preface.tex) are generic template files that are included in DO-178A documents and all the information for those templates is supplied by the document author in the document driver. When the document driver is LaTeXed, it uses the author's definitions to fill in the symbols that are part of the three generic templates as it automatically includes the report_cover, signature page, and preface.

11. Accuracy Requirements

The requirements for the accuracy of certain variables in the *GCS Development Specification* is described in this programmer instruction. A minor problem discovered during the analysis is also described.

Explanation of Requirements

The numerical data for the requirements will replace the TBD entries in the data requirements dictionary. The exact numbers will be issued as a specification modification, and the data will be incorporated into a future release of the *GCS Development Specification*. The following is an explanation of the numerical data given in the specification modification.

The accuracy requirements are numerical values which describe a maximum allowable amount of "distance" between two values. The values that will be compared will be a value of the same variable at the same point in time for at least two different implementations. The accuracy requirement for a variable x is symbolically designated as δ_x. Each variable in the data requirements dictionary has its own accuracy requirement. The "distance" between values is described in terms of *relative* error; that is, the variable x in one version (x_1) must be within δ_x of the same variable in the second version, x_2, according to the following computation:

$$\frac{|x_1 - x_2|}{|x_2|} \leq \delta_x$$

The accuracy requirements can also be referred to as a required number of significant digits. That is, if δ_x is written in scientific notation, then x_1 approximates x_2 to t significant digits if t is the largest non-negative number such that $\delta_x < 5 \times 10^{-t}$.

Description of Possible Problem

In FORTRAN, the variable types of all variables in mathematical operations are upgraded to match the one with the "highest" variable type. (The order from lowest to highest is: logical*1, logical*2, integer*2, integer*4, real*4, real*8, complex*8 and complex*16.) While all other upgrades perform as expected, the upgrade from real*4 to real*8 adds nine random digits after the last significant digit.

Example: the real*4 number 0.5050000 becomes the real*8 number 0.5049999952316284

Because there are no real*4 data types in the data dictionary in the *GCS Development Specification*, this should pose no problem during coding. However, each programmer should be aware that if real*4 data types are used and mixed with real*8 data types, there is a potential problem for perturbing the results enough to conflict with the accuracy requirements.

12. Programmer Responsibilities

12.1. Introduction

This programmer instruction identifies the different areas of programmer responsibility with respect to testing, software quality assurance, and configuration management. It is intended to be an overview of these procedures. All procedures are listed in more detail in the *Software Verification Plan for GCS*, the *Software Quality Assurance Plan for GCS*, and *GCS Configuration Management Plan.*

12.2. Design Review

- Once the design is complete, the programmer and tester together decide when the design is ready for review.
- The programmer makes one copy of the team*work* design, and the copy and original are circulated to the review team (user, tester and SQA representative) 48 hours before the review.
- The programmer starts the review by giving a brief overview of the design.
- The programmer leads the team through the design by explaining the design diagrams, interpreting the design, and answering any questions about it.
- The design will be checked against the GCS Requirements Traceability Matrix and GCS Design Review Checklist, which are in an appendix to the *Software Verification Plan for GCS.*
- Once the review is finished, the programmer is responsible for completing problem reports relating to the design and changing the design appropriately.

12.3. Code Review

- After the code has been written, compiled, and linked without error, a version of the code will be put under configuration management. Before the Code Review, NO PART of the code should be executed.

- The programmer and tester decide together when the code is ready for review. The code will be checked against the GCS Requirements Traceability Matrix and GCS Code Review Checklist, which are in an appendix to the *Software Verification Plan for GCS.* The code will also be checked against the design.

- The programmer gives a copy of the selected units to each member of the review team (user, tester and SQA representative) at least 24 hours before the review so that the review team members can look over the units prior to the review.

- The programmer starts the review by giving an overview of each code unit. The overview should include the function of the unit, where it fits in the subframe and the relationship of the unit to the design.

- The programmer leads the team through the code by reading it aloud, line by line, interpreting the code and answering any questions about it.

- Once the review is finished, the programmer is responsible for completing problem reports relating to the code or design, and changing the code or design appropriately.

12.4. Unit Testing

The programmer is fully responsible for the unit testing. While a brief outline of the procedure is listed below, the programmer is encouraged to read the Unit Testing section of the *Software Verification Plan for GCS.*

- The User/Analyst will provide the programmer with one set of run parameters which the programmer can use during testing.

- The programmer will have his/her own CMS library during unit testing. Each entry in the CMS library should correspond to a unit test case.

- The programmer builds and executes at least three test cases per unit for a minimum of 20 test cases per subframe. Each test case must be logged on GCS Module Test Log forms, which are found in an appendix to the *Software Verification Plan for GCS.* There should be one test log per code unit.

- The programmer must re-execute any test case which found an error until all test cases execute correctly.

- If more than 20 lines of code have been added or modified, the code must go through another code review.

- Following Unit Testing, there will be a test completion review which will check that the correct number of tests were performed, all tests were logged, and all problems were corrected.

12.5. Subframe, Frame and System Testing

- The tester performs the subframe, frame, and system testing.
- The programmer fixes any problems found after a complete test run and completes the accompanying GCS Problem Report Form for each error. If an error is traced to the design, a problem report form is filled out and the programmer fixes the design error.
- If more than 20 lines of code have been added or modified during one code fix, the code must go through another code review.
- After the programmer fixes all problems, the tester will rerun the test cases to insure that all test cases run correctly.
- Once all fixes are complete, the programmer submits the corrected code to SQA for entry into CMS.
- A test completion review is held to check that all problems were corrected and are appropriately marked in the code.

References

[1] RTCA Special Committee 152. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178A, Radio Technical Commission for Aeronautics, March 1985.

[2] George B. Finelli. Results of software error-data experiments. In *AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference*, Atlanta, GA, September 1988.

[3] Alan Roberts, Don Rich, and John Pierce. *Internal Document : VMS FORTRAN Code Generation Guidelines.* Software R & D Department, Center for Digital Systems Research, Research Triangle Institute, Research Triangle Park, NC, June 1986.

[4] Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification.* Dorset House Publishing Company, New York, New York, 1987.

[5] Tom DeMarco. *Structured Analysis and System Specification.* YOURDON Inc., 1133 Avenue of the Americas, New York, NY 10036, 1978.

[6] Paul Ward and Steven Mellor. *Strucured Development for Real-Time Systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1985.

[7] E. Presson. *Software Test Handbook/Software Test Guidebook.* RADC-TR 84-53, Rome Air Development Center, March 1984.

[8] Leslie Lamport. LaTeX: *A Document Preparation System.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1986.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-102721		

4. Title and Subtitle	5. Report Date
GCS Programmer's Manual	December 1990
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Douglas S. Lowman, B. Edward Withers, Anita M. Shagnea, Leslie A. Dent, and Kelly J. Hayhurst	
	10. Work Unit No. 505-66-21-03

9. Performing Organization Name and Address	11. Contract or Grant No.
NASA Langley Research Center Hampton, VA 23665-5225	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

Douglas S. Lowman, B. Edward Withers, Anita M. Shagnea, and Leslie A. Dent: Research Triangle Institute, Research Triangle Park, North Carolina.
Kelly J. Hayhurst: Langley Research Center, Hampton, Virginia.

16. Abstract

This document describes a variety of instructions to be used in the development of implementations of software for the Guidance and Control Software (GCS) project. This document fulfills the Radio Technical Commission for Aeronautics RTCA/DO-178A guidelines, "Software Considerations in Airborne Systems and Equipment Certification" requirements for document No. 4, which specifies the information necessary for understanding and programming the host computer, and document No. 12, which specifies the software design and implementation standards that are applicable to the software development and testing process. Information on the following subjects is contained in this paper: activity recording, communication protocol, coding standards, change management, error handling, design standards, problem reporting, module testing logs, documentation formats, accuracy requirements, and programmer responsibilities.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Guidance and Control Software (GCS) Communication Protocol Programmer Instruction Design and Coding Standards	Unclassified - Unlimited Star Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	28	A03